*Natural Language Procesing with Python Chapter 2*

Various Corpora

|  |  |
| --- | --- |
| Problem | Solution |
| Get a list of texts (returns a list) | import nltk  nltk.corpus.gutenberg.fileids() # includes 'austen-emma.txt' |
| Get the number of words in a specific text | emma = nltk.corpus.gutenberg.words('austen-emma.txt') #words function of gutenberg  type(emma) >>> <class 'nltk.corpus.reader.util.StreamBackedCorpusView'>  len(emma) >> 192427 |
| Simplify Above | from nltk.corpus import gutenberg  len(gutenberg.words('austen-emma.txt') |
| Number of characters | len(gutenberg.raw('austen-emma.txt')) #this includes spaces |
| Number of words | len(gutenberg.words('austen-emma.txt')) |
| Number of sentences | len(gutenberg.sents('austen-emma.txt')) |
| Example of sentence structure | sentences = gutenberg.sents('austen-emma.txt')  sentences[:3]  [['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']'], ['VOLUME', 'I'], ['CHAPTER', 'I']] |
| Length Longest Sentence | longestLength = max([len(s) for s in sentences]) # uses sentences abvoe |
| Actual Longest Sentence | [s for s in sentences if len(s) == longestLength] #can return multple entries |
| Webtext Corpus | About 5 different chatroom corpus |
| Print the first 50 characters | webtext.raw('firefox.txt')[:50] |
| Brown Corpus | First million word English corpus. 500 sources. Categoriesed into genres. |
| Get a list of categories in Brown | from nltk.corpus import brown  brown.categories() |
| Get the words in a category | newsWords = (brown.words(categories = 'news'))  len(newsWords) >> 100554 |
| What is “stylistics” ..? | The systematic differences between genres |
| Get a frequency distribution of Modal Verbs in news | import nltk  modals = ['can', 'could', 'may', 'might', 'must', 'will']  fdist = nltk.FreqDist([w.lower() for w in newsWords if w.lower() in modals])  print(fdist) >>> <FreqDist: 'will': 389, 'can': 94, 'may': 93, 'could': 87, 'must': 53, 'might': 38> |
| Compare multiple genres using modal verbs | Construct a cross tabulation of the 6 verbs against the N genres. A 6 x N matrix. |
| What is lingustic modality..? | Is what allows speakers to attach expressions of belief, attitude and obligation to statements |
| The Reuters Corpus | This has over 10K news articles classified into 90 topics. News articles can span multiple topics. (see page 45) |
| Innaugural Address Corpus | The corpus is a collection of 55 texts. One for each presidential address. |
| Get the first 3 file ids | inaugural.fileids()[:2] >>> '1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', |
| Extract the year from the id | [id[:4] for id in inaugural.fileids()] |
| Annotated Corpora | Many text corpora contain linguistic annotations representing part of speech tags, named entities…etc. There are chunking corpora, pronouncing dictionaries, gazetterrs. (list on page 47) |
| Types of corpus | Isolated (no organisation), Categorised (non-overlapping), Overlapping, Temporal (such as innaugural) |
| Loading your own corpus | from nltk.corpus import PlaintextCorpusReader  root = r"/Users/zurich/Google Drive/NLTK-2.0/rudd"  wordlists = PlaintextCorpusReader(root, 'RuddsSpeech.txt')  wordlists.fileids() >> ['RuddsSpeech.txt'] |
| Converting corpus to Text and running collocations() | words = wordlists.words()  textObject = nltk.Text(words)  textObject.collocations() |

Conditional Frequency Distributions

|  |  |
| --- | --- |
| Problem | Solution |
| What is a Conditional Frequnecy Distribution..? | This is a freqency distribution – one for each condition. A condition can be a genre. |
| Data structure for Conditional Frequency Distribution | The CFD needs to pair each event with a condition. An event can be a word. The data structure is a list of pairs: [(‘news’,’the’), (‘news’,cat), (‘news’,sat), (‘news’,on), (‘news’,’the’),] The form is (condiiton, even). |
| Idiom 1(get words for a category) | from nltk.corpus import brown  words = [w for w in brown.words(categories='news')]  len(words) >>> 100,554  words[:10] >>> ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of']  len(set(words)) >>> 14,394 |
| Idiom 2 | [genre for genre in ['news', 'categories']] >>> ['news', 'categories'] |
| Combining Idioms | wordPairs = [(genre, word) for genre in ['news','romance'] for word in brown.words(categories=genre)]  #The outer (green) loop is passed to the red loop. The red loop runs the first time with ‘news’ then runs again with ‘romance’ |
| Using the conditional frequency distribution function | from nltk.probability import ConditionalFreqDist  cfd = ConditionalFreqDist(wordPairs)  cfd.conditions() >>> ['news', 'romance']  cfd[‘news’] >>> <FreqDist with 100554 outcomes> <class 'nltk.probability.FreqDist'> |
| Getting the word lengths from different languages | from nltk.corpus import udhr  languages = ['Chickasaw', 'English', 'German\_Deutsch', 'Greenlandic\_Inuktikut', 'Hungarian\_Magyar', 'Ibibio\_Efik']  wordPair = [(lang, len(word)) for lang in languages for word in udhr.words(lang + '-Latin1')]  cfd = ConditionalFreqDist(wordPair)  cfd.tabulate(conditions=['English', 'German\_Deutsch'], samples=range(10), cumulative=True) |
| Results of the above | 0 1 2 3 4 5 6 7 8 9  English 0 185 525 883 997 1166 1283 1440 1558 1638  German\_Deutsch 0 171 263 614 717 894 1013 1110 1213 1275 |

The Structure of the Frequence Distribtion Object *(Continued)*

|  |  |
| --- | --- |
| Problem | Solution |
| Read in Genesis | import nltk  text = nltk.corpus.genesis.words('english-kjv.txt') >>> ['In', 'the', 'beginning', 'God', 'created', 'the', 'heaven', 'and', 'the', 'earth'] |
| Convert to bigrams (from above) | bigrams = nltk.bigrams(text)  bigrams[:10] >>> [('In', 'the'), ('the', 'beginning'), ('beginning', 'God'), ('God', 'created'),…… |
| From the list of two element tuples above we can use the FreqDist object | cfd = nltk.ConditionalFreqDist(bigrams)  # in the above, the first element in the tuple is the condition.. such as ‘In’, ‘the’, ‘beginning’  # so these are the keys() for the resultant dictionary. |
| Get a dictionary (like) object | testObject = cfd['living']  type(testObject) >>> <class 'nltk.probability.FreqDist'> ***[Subclass of dictionary I think]*** |
| Show some dictionary properties: | testObject.keys() >>> ['creature', 'thing', 'substance', ',', '.', 'soul']  testObject['creature'] >>> 7  # can get the same as above by:  cfd['living']['creature'] >> 7 |
| Random Text (generate\_model) | def generate\_model(cfdist, word, num=15):  for i in range(num): # [0,1,2….14]  print word, #prints word  word = cfdist[word].max() #prints cfdist[word].max() and assigns to word  # call the function above  generate\_model(cfd, 'living')  # results in: living creature that he said , and the land of the land of the land |
| Explantion of above | cfd['living'].max() #>>> creature  cfd['creature'].max() #>>> that  cfd['that'].max() #>>>he |

More Python – Reusing Code

|  |  |
| --- | --- |
| Problem | Solution |
| Run a text file | In Idle goto file > new can create new file. Then save (give it a name) then goto F5 and press run. |
| word[-1] == word[-1:] | word = ‘wish’ >>  word[-1] >>> h  word[-1:] >> h |
| Modules (create and use) | To create. Just save a test file with the “py” extension. Assume it contains def fnTestFunction(): And assume it is saved as testModule.py  To use:  >>>from testModule.py import fnTestFunction()  >>>fnTestFunction() # |
| Module, Package and Library | Module is a collection of related functions and variables  Package is a collecton of related modules  Library is a collection of related Packages |

Lexical Resources

Lexical resouces are secondary to texts. But they are usually created with the help of texts. For example a vocabulary of words is a simple resources. A **lexical entry** consists of a **headword** (also known as a **lemma**) along with additional information such as the part-of-speech and the sense definition. Two distinct words having the same spelling but different definitions are colled **homonyms** (such as saw (past tense of see), and saw (cutting instrument). *[Dictionary defintion of a lemma: a heading indication the subject or argument of a literary composition or annotation].*

Wordlist Corpora *(English Words, Stopwords, Christian Names)*

|  |  |
| --- | --- |
| Problem | Solution |
| Get a list of **English words** | nltk.corpus.words.words() |
| Function that returns words that are not English (unusal words) | def fnUnusualWords(text):  textVocab = set(w.lower() for w in text if w.isalpha()) #set all words to lower case if alpha  englishVocab = set(w.lower() for w in nltk.corpus.words.words()) #  unusual = textVocab.difference(englishVocab) #set difference  return sorted(unusual) # |
| Get a list on English **stopwords** | from nltk.corpus import stopwords  stopwords.words('english') |
| Puzzle words on page 61. We have a puzzle with three columns and three rows. The letters in the puzzle in left to right; top to bottom are: E, G, I, V, R, V, O, N, L. The length of the word must be greater than 6. R must be in the word. And ALL letters in the candidate word must be in the 9 letters. Also, a letter cannot be used more than one. So, V is repeated twice above and therefore it can be used twice in a solution. | |
| Create a frequency distribution of the letters | puzzleLetters = nltk.FreqDist('egivrvonl')  puzzleLetters.items() >>> [('v', 2), ('e', 1), ('g', 1), ('i', 1), ('l', 1), ('o', 1), ('n', 1), ('r', 1)] |
| Test to see if the elements of one list are ALL contained | testLetters = nltk.FreqDist('region')  testLetters <= puzzleLetters # results in true  testLetters = nltk.FreqDist('region')  testLetters <= puzzleLetters # results in false  testLetters = nltk.FreqDist('regionvv')  testLetters <= puzzleLetters # results in true v can be repeated. |
| Run the exercise: | >>> wordList = nltk.corpus.words.words()  >>> puzzleLetters = nltk.FreqDist('egivrvonl')  >>> for word in wordList:  if nltk.FreqDist(word) <= puzzleLetters and len(word) >= 6:  print(word) |

|  |  |
| --- | --- |
| Problem | Solution |
| List of male / female names | names = nltk.corpus.names  names.fileids() >>> ['female.txt', 'male.txt']  lstFemale = names.words('female.txt')  lstFemale[300:315] >>> ['Annetta', 'Annette', 'Anni', 'Annice', 'Annie'…….. |

Pronouncing Dictionary

|  |  |
| --- | --- |
| Problem | Solution |
| Show the basic structure | entries = nltk.corpus.cmudict.entries()  len(entries) >>> 133737  entries[100:102] >>> [('abdication', ['AE2', 'B', 'D', 'IH0', 'K', 'EY1', 'SH', 'AH0', 'N']), ('abdnor', ['AE1', 'B', 'D', 'N', 'ER0'])]  The above codes are from **Arpabet There are a list of entries. Each entry consists of a two element tuple. The first element is the word and the second element is a list of the proncouncation elements.** |
| Show the words that sound as though they start with *‘p’* and end with *‘t’* | for word, pron in entries: # entries contains two elements  if len(pron) == 3:  p1, p2, p3 = pron # pron contains three elements  If p1 == 'P' and p3 == 'T':  print word >>> pate, pot….. |
| Find words that end with “icks” | [word for word, pron in entries if word.endswith('icks')]  # results in: 'bricks', 'broomsticks', 'buicks', 'candlesticks', 'chicks', 'chopsticks', 'clicks'…….. |
| Find pronouncation for the above | [pron for word, pron in entries if word.endswith('icks')] |
| Get last three elements for above | [pron[-3:] for word, pron in entries if word.endswith('icks')]  # [['IH0', 'K', 'S'], ['IH1', 'K', 'S'], ['IH2', 'K', 'S'], ['IH0', 'K', 'S']….. |
| Get words that sound as though they end in “icks” | [word for word, pron in entries if pron[-3:] == syllable] |
| Get words that end in “m” but sound like they end in “n” | Not shown on page 64 |
| Get the pronouncation for a specific word | pronoun = nltk.corpus.cmudict.dict()  pronoun['bud'] >>> [['B', 'AH1', 'D']] |

Comparative Wordlist

|  |  |
| --- | --- |
| Problem | Solution |
| Get the basic sucker working | from nltk.corpus import swadesh  swadesh.fileids() # returns ['be', 'bg', 'bs', 'ca', 'c…….  swadesh.words('en') #returns ['I', 'you (singular), thou', |
| Build a dictionary from a list of sequences | markDict = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])  markDict['sape']  4139 |
| Create list of sequences between two languages | fr2en = swadesh.entries(['fr', 'en'])  fr2en[:15] #returns [('je', 'I'), ('tu, vous', 'you (singular), thou'), ('il', 'he')……] |
| Make a dictionary from the above | translate = dict(fr2en)  translate['je'] >> I |
| Append a dictionary to another | dict1 = {'Name': 'Zara', 'Age': 7}  dict2 = {'Sex': 'female' }  dict1.update(dict2)  dict1  {'Age': 7, 'Name': 'Zara', 'Sex': 'female'} |
| Language comparison | languages = ['en' , 'de', 'nl', 'es', 'fr', 'pt', 'la']  marklist = swadesh.entries(languages)  marklist[1] # ('you (singular), thou', 'du, Sie', 'jij, je, U', 't\xc3\xba, usted', 'tu, vous', 'tu, voc\xc3\xaa', 'tu')  marklist[45] # ('bird', 'Vogel', 'vogel', 'ave, p\xc3\xa1jaro', 'oiseau', 'ave, p\xc3\xa1ssaro', 'avis') |
| ***Shoebox and toolbox lexicons*** | To be done in later chapters. |

Wordnet

|  |  |
| --- | --- |
| Problem | Solution |
| Definition of a synset | A set of words that are interchangeable in some context without changing the truth value of the preposition in which they are embedded. |
| In word net a synset is an identifer for a meaning. | In English, words can have one or more meanings. Therefore, they can have one or more synset identifiers. |
| Getting the synset (identifier) | from nltk.corpus import wordnet as wn  wn.synsets('motorcar') >> [Synset('car.n.01')] |
| Getting all the words associated with a specific meaning *(not the argument is a synset identifier)* | wn.synset('car.n.01').lemma\_names  ['car', 'auto', 'automobile', 'machine', 'motorcar'] |
| Getting a synset definition | wn.synset('car.n.01').definition >>> 'a motor vehicle…… |
| Getting a synset example | wn.synset('car.n.01').examples >>> ['he needs a car to get to work'] |
| Getting lemma identiers | wn.synset('car.n.01').lemmas >>> [Lemma('car.n.01.car'), Lemma('car.n.01.auto'),……. |
| Getting a synset for a lemma *(should only be one)* | wn.lemma('car.n.01.automobile').synset >>> Synset('car.n.01') |
| There can be multiple synsets (meanings ) for a word | wn.synsets('car') >>>> [Synset('car.n.01'), Syns………….  # such as cable car, motorcar…. |
| Hyponyms (more specific) | wn.synset('car.n.01').hyponyms() # returns a list including words like “hatchback”, “racer” |
| Hypernyms (more general) | wn.synset('car.n.01').hypernyms() >> [Synset('motor\_vehicle.n.01')] |
| Paths. There can be multiple because sometimes there two meanings | wn.synset('car.n.01').hypernym\_paths() #returns paths from most general (i.e. ‘entity’ ending with ‘car.n.01’) |
| What are meronyms..? | They are the parts of an object. There are part meronyms and substance meronyms. Examples below. |
| Part Meronym | wn.synset('tree.n.01').part\_meronyms() >> Synset('stump.n.01'), Synset('trunk.n.01'), |
| Substance Meronym | wn.synset('tree.n.01').substance\_meronyms() [Synset('heartwood.n.01'), Synset('sapwood.n.01')] |
| Holonyms *(what collection object is a member of)* | wn.synset('tree.n.01').member\_holonyms() >>> [Synset('forest.n.01')] |
| Entailment *(eating entails chewing)* | wn.synset('eat.v.01').entailments() >>> [Synset('swallow.v.01'), Synset('chew.v.01')] |
| Antonyms between lemmas | wn.lemma('supply.n.02.supply').antonyms() >>> [Lemma('demand.n.02.demand')] |
| Semantic similarity | *Wordnet can look up the tree for the lowest common hpernym. This is shown below:*  right = wn.synset('right\_whale.n.01')  orca = wn.synset('minke\_whale.n.01')  right.lowest\_common\_hypernyms(minke) >>> [Synset('baleen\_whale.n.01')] |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |